User HMI for PA Control

**Subject for technical modifications and errors*

HISTORY 4

INTRODUCTION 4

REFERENCES 4

User HMI Example 5

CONFIGURATION LEVEL 5

General 5
How To Create Standalone Application 6
How To Add New Soft Key Level in HMI 7
Viewer Types 7
How To Define More (Less) Then 8 Soft Keys 8
How To Define User Hard Keys 8
Visualization Of PLC Program 8

EXPERT LEVEL 8

General Communication With CNC Server 8

Communication Between HMI and User Applets 12

Java Applets 15
Sun Java vs. Microsoft Java (Windows version only) 15
Java Sources 16
Realization of Command Interface to CNC Server 17
Getting of Data From CNC Server on Request With Automatic Updating Mode 19
How to Build Java Archive 19

Development of ActiveX Controls 20
HMI Standard Applet “activex” as Container of Control 20
ActiveX Examples 21

DEVELOPMENT USER SPECIFIC OBJECTS (APPLETS, FUNCTIONS) BY QT TOOLKIT 22

OTHERS 22

COMPARISON OF TECHNOLOGIES 24

FAQ 24

How Many Soft Key Levels Are Possible? 24

Subject for technical modifications and errors

What's Happened With The Focus Control In Windows, When Will It Change From HMI To Applets? 24

Can Java Applets or ActiveX Access The Soft Keys? 25

How Can An OEM Applet (Java, ActiveX) Work With Selected HMI Fonts, Colors? 25

I search for a possibility to get soft key information in an additional applet 25

Java applet is shown by Microsoft Internet Explorer but not displayed inside HMI 26

**Subject for technical modifications and errors*

History

Revision	Date	Comments
01	07.04.2003	Document created
02	12.01.2004	Completely redesigned. Supplied by examples for ActiveX and Java technologies
03	17.05.2004	New FAQ (I search …) added
04	02.08.2004	New FAQ (Java applet is shown by Microsoft Internet Explorer but not displayed inside HMI) added
05	06.10.2004	User hard keys, Visualization added
06	17.11.2004	• HMI can overwrite title, tool tip and icon of user soft keys, when corresponding <pool> comes • HMI handle macro command when corresponding <pool> comes
07	08.02.2007	New checkForClose() method for ActiveX (since HMI kernel version 1.24.10)

Introduction

This is user manual for customizing of Human Machine Interface (HMI) of CNC.

HMI is flexible and open application. There are two levels of HMI customizing:

- Configuration level
- Expert level

Selection of level depends on user's requirements and experience.

Configuration level is simplest way of customizing. It demonstrates flexibility of HMI. By this way user can reconfigure HMI layouts and make own assignments for hard and soft key functions. Configuration level is shortly described in current manual.

Expert level reflects openness of HMI. Experienced users (are familiar with programming) can customize HMI on this level. Level is recommended when there are no more possibility to solve user's requirements via reconfiguration of HMI. Detailed description of HMI customizing on expert level is subject of current manual.

Only HMI 1.18.1 and later supports all features.

Java and ActiveX sections of documentation is applied for Windows version only!

References

Manual refers to following documents:

- SD001EN CNC Server (TCP)
- SD002EN HMI Setup and Configuration Guide
- UD017EN User Visualization By HMI

*Subject for technical modifications and errors

User HMI Example

Standard HMI distributive may include examples. Please, contact to manufacturer to get examples. All examples (included source code) are located in **userhmi** folder. Examples are applied to Windows HMI version 1.16 and higher.

Structure of folder:

- **activex** – Microsoft Visual Studio C++ projects for ActiveX controls
- **cfg** – additional specific configuration files to run User HMI application
- **images** – additional images
- **doc** - documentation
- **j2sdk** – original Java SDK, downloaded from java.sun.com
- **java** – Java applets
- **java-dde** - Microsoft Visual Studio C++ project to build Java-DDE library. Library provides DDE interface for Java applets
- **plc** – PLC program project for Visualization
- **xml** - additional specific xml files to run User HMI application

Run **prepare.bat** to prepare User HMI application. Batch file copies all required files to **hmi** working directory and registers ActiveX components. Note: standard HMI should be installed before!

User HMI application can be started by command line (see **start.bat**):

hmiframe.exe" -i:user –fi:userframe

or by script **start.bat**

Configuration Level

General

User doesn't need additional tool and experience in programming to customize standard HMI on this level. All can be done by simple text editor and graphic editor which are included into operating system. See SD002EN "HMI Setup and Configuration Guide" for details.

User can:

- Change shape of buttons, icons (manually by graphic editor)
- Change fonts, number of hard keys and soft keys, location of tool bars (directly in HMI Setup dialog or via ini files)
- Assign (reassign) function for each hard or soft key (most of soft keys can be assigned in HMI Setup dialog, complete assignment is carried out via cfg files)
- Assign layout (from existing list) for each SK level (via cfg files)
- Create own layout or change existing ones as required (via XML files). Margins, background color, size and position, fonts, whether border and title is present or not and other information are specified in XML files.

All configuration files and XML files are compatible for both Windows and Linux versions.

How To Create Standalone Application

It is necessary to prepare two files (ini and cfg) by any text editor. More details are described in SD002EN “HMI Setup and Configuration Guide”. Let to see some files to run Path Graphic Display as standalone aplpication.

pg.ini

```
[ini]
swvisible=0
title=Path Graphic
nextinstance=0
cfg=pg
[user]
icon=pa
[pg]
on=1
```

pg.cfg

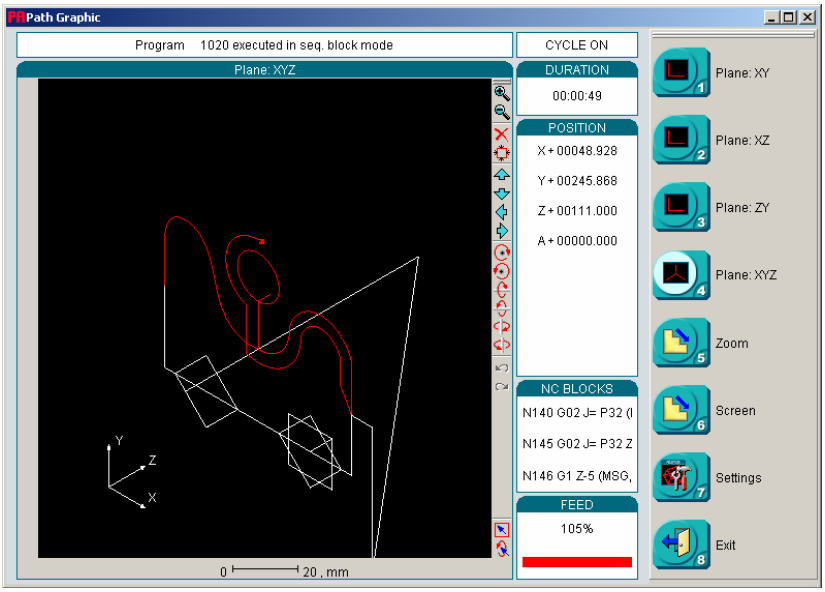
```
[main]
00000000=pg.htm$plain
00000001=pgplanexy:hmi
00000002=pgplanexz:hmi
00000003=pgplanezy:hmi
00000004=pgplanexyz:hmi
00000005=go#zoom:hmi
00000006=go#screen:hmi
00000007=pgsettings:hmi
00000008=exit
;sk5, zoom
50000001=pgzoomin:hmi
50000002=pgzoomout:hmi
50000003=pgup:hmi
50000004=pgdown:hmi
50000005=pgleft:hmi
50000006=pgright:hmi
50000008=back
;sk6, screen
60000002=pgfitin:hmi
60000004=pgclear:hmi
60000008=back
```

Now we can start new HMI application by command line string

hmiframe.exe -i:pg

*Subject for technical modifications and errors

and see Path Graphic display



How To Add New Soft Key Level in HMI

New soft key level can be added manually via configuration file. Details are described in SD002EN “HMI Setup and Configuration Guide”. User HMI example demonstrates how to add new levels.

Maximum 7 levels can be configured directly. Using of complicated functions (i.e. Editor of machine parameters into HMI) allows to get unlimited number of levels indirectly.

Note:

- HMI reports current level to PLC. Only seven upper levels can be recognized by PLC. High four bits of double word (32 bits) represent hard key level (0 – level of first hard key, 1 – level of second hard key, …). Next four bits correspond first sublevel of soft keys (1 - first soft key, 2 – second soft key, …) etc.
- HMI (master) reports current level to user applets (Java, ActiveX) by communication object <pool> in similar manner.

Viewer Types

Various types of viewers are demonstrated in User example (see How To Add New Soft Key Level in HMI).

- Soft key 1 (Plain viewer) opens new level where Qt applets are shown. One of Qt applet (top left) is Microsoft Internet Explorer (Windows). The applet “web” shows web page because that is specified by <PARAM name='url' value='www.trolltech.com'/>. It is possible to specify file path for “web” applet, if ‘file’ is used instead of ‘url’
- Soft key 2 (Web viewer) opens new level where Microsoft Internet Explorer is viewer. Viewer shows web.html file. Animated picture is shown on page

*Subject for technical modifications and errors

- Soft key 3 (Textbrowser viewer) opens new level where Qt text browser is viewer. Viewer shows textbrowser.html file. Content of this file is specification of structuring tags accepted by text browser. Specification was extracted from Qt documentation

How To Define More (Less) Then 8 Soft Keys

Example demonstrates 10 soft keys. To define 10 keys:

- Specify sknumber=10 in frame **ini** file
- Create icons for on and off key statuses (sk9off.gif, sk9on.gif, sk10off.gif, sk10on.gif)
- Specify additional key codes for new keys in [localkeys] section of frame **ini** file
  softkey9=alt+9
  softkey10=alt+0
- Use new levels for key definitions in **cfg** configuration file (for example, 0000000a=exit)

How To Define User Hard Keys

Example demonstrates two user defined hard keys. To define user hard keys:

- Define user hard keys in [hardkeys] section of **cfg** configuration file
  [hardkeys]
  00000001=hardkey#userkey1$Applets$HMI, Java applets and ActiveX controls$user
  00000002=hardkey#userkey2VisPLC visualization$user
  Here name and tool tip of each user hard key are specified. Both keys will use the same icons **user** (user16.gif)
- Define own section for each hard key level. There are two sections in **cfg** configuration file: [userkey1] for first user hard key and [userkey2] for second

Visualization Of PLC Program

User visualization system provides customer by easy and powerful way to display state of PLC variables and exchange data between HMI applets and PLC. See UD017EN User Visualization By HMI documentation for details. Visualization applets are running in **userhmi** when second hard key selected.

Note: PLC runtime should be activated and special PLC program from userhmi/plc directory loaded!

Expert Level

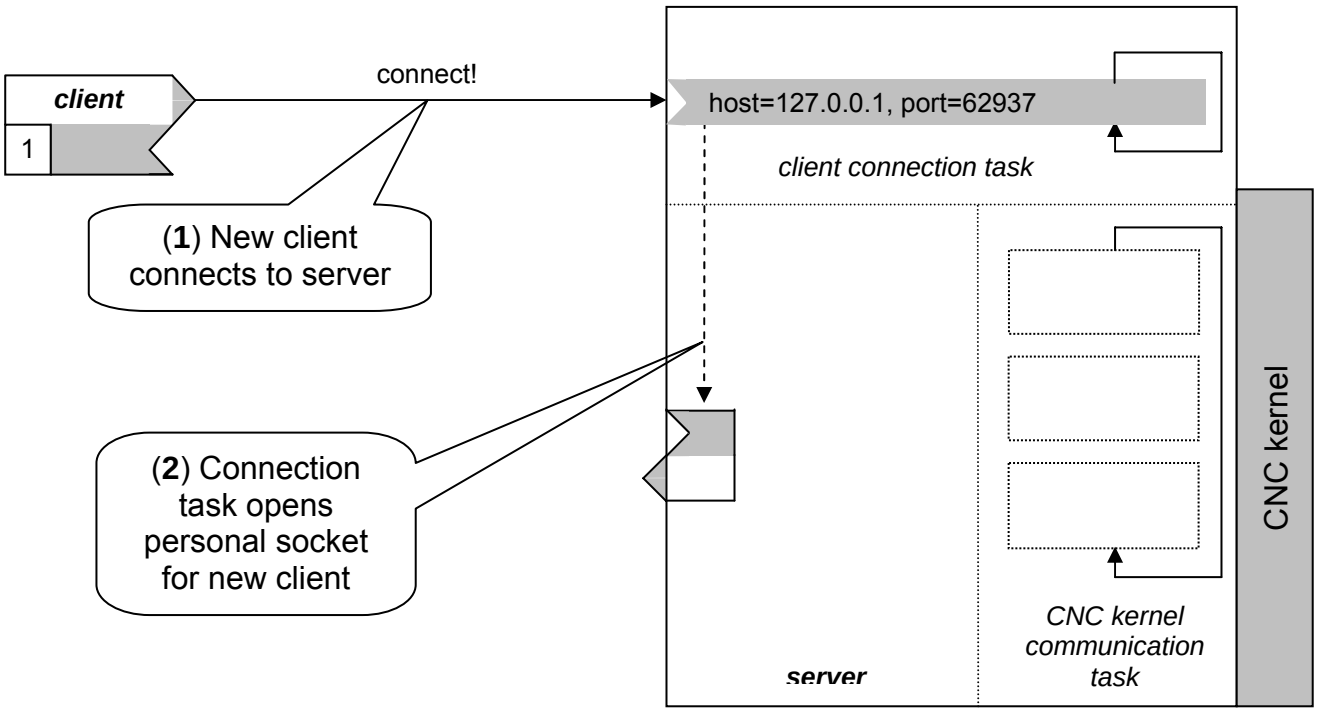
General Communication With CNC Server

One of important thing for understanding of expert level is communication with CNC server. When HMI or other clients run they don't operate directly with CNC kernel but through special intermediate server (TCP/IP). Server provides clients by full information about actual CNC state. Of course, clients can change a state of CNC if required. Server has access to CNC data via shared memory.

New user's applets (ActiveX or Java) can install independent communication channel to CNC server. Open XML protocol is used for data exchange. Detail description of protocol is documented in specification: SD001EN CNC Server (TCP).

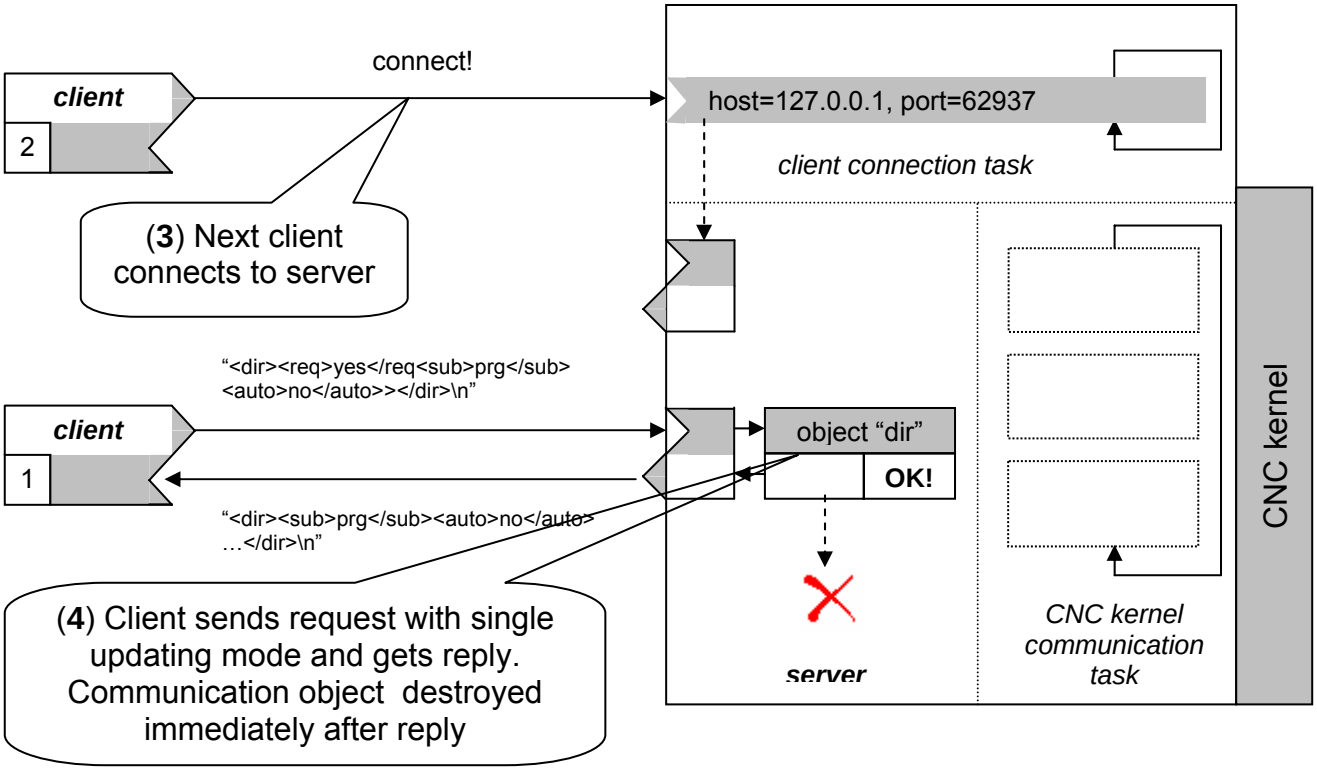
Client has to know IP address of server and port to create connection. CNC server handles several ports for different functionality (main server, file server, path graphic server etc). Main server provides client by general CNC functions. Default number of main server port is 62937.

Server listens port by internal client connection task (thread). When new client detected, then server creates personal socket. Below diagram shows connection of first client.



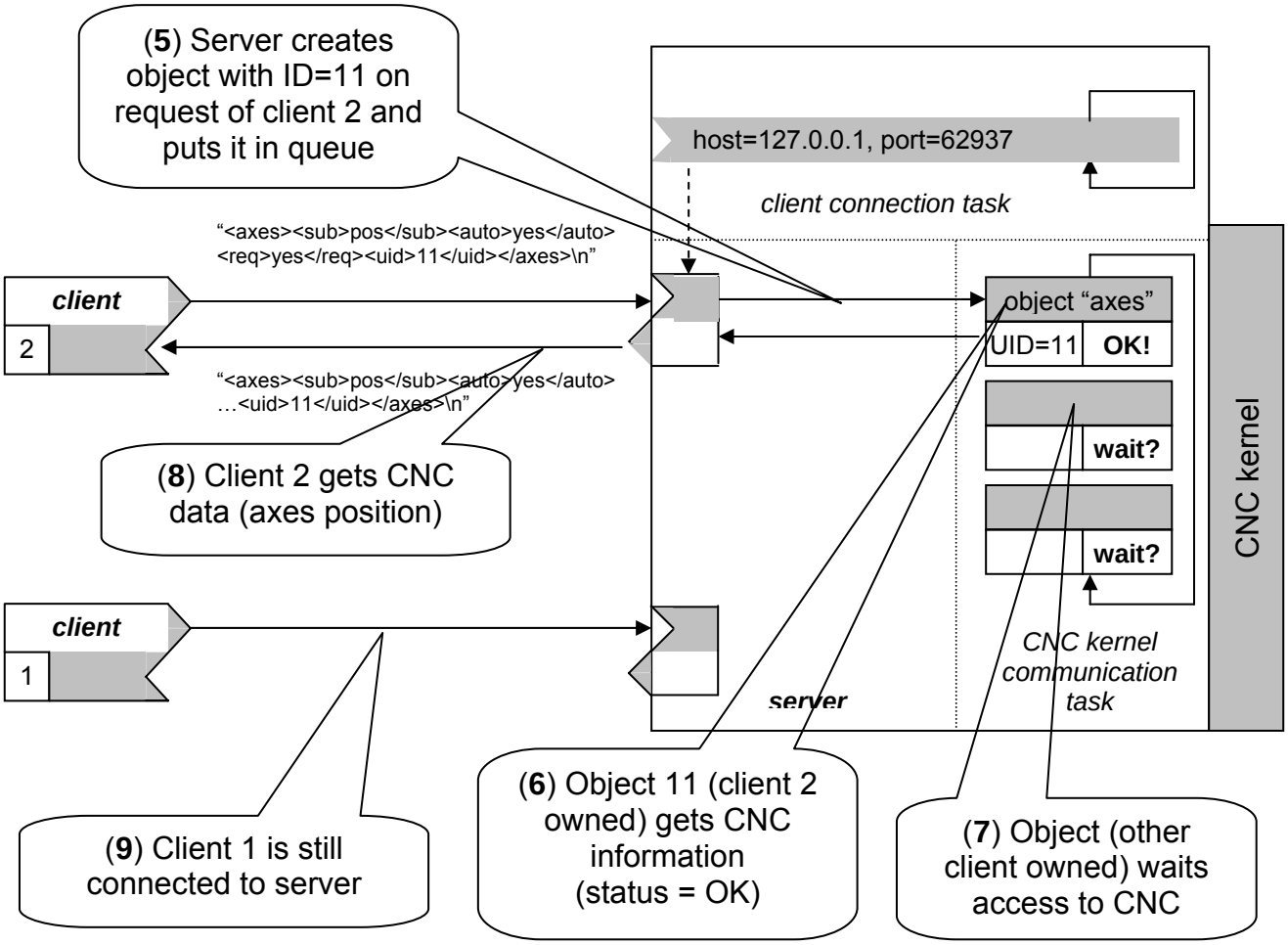
Client sends request to server as XML string terminated by '\n'. When server gets message from client then it creates internal communication object corresponding to request. If request is specified by client in **single updating mode**, then server doesn't put corresponding communication object in queue of requests. Server just gets CNC data and sends reply to client. Communication object destroyed immediately after that.

*Subject for technical modifications and errors



If request is specified by client in **automatic updating mode**, then server puts corresponding communication object in queue of requests. Special communication task handles object in queue. Each object waits when CNC information will be available and sends reply to client (owner of object). If object can't get CNC information now, it waits next pass of communication task.

*Subject for technical modifications and errors



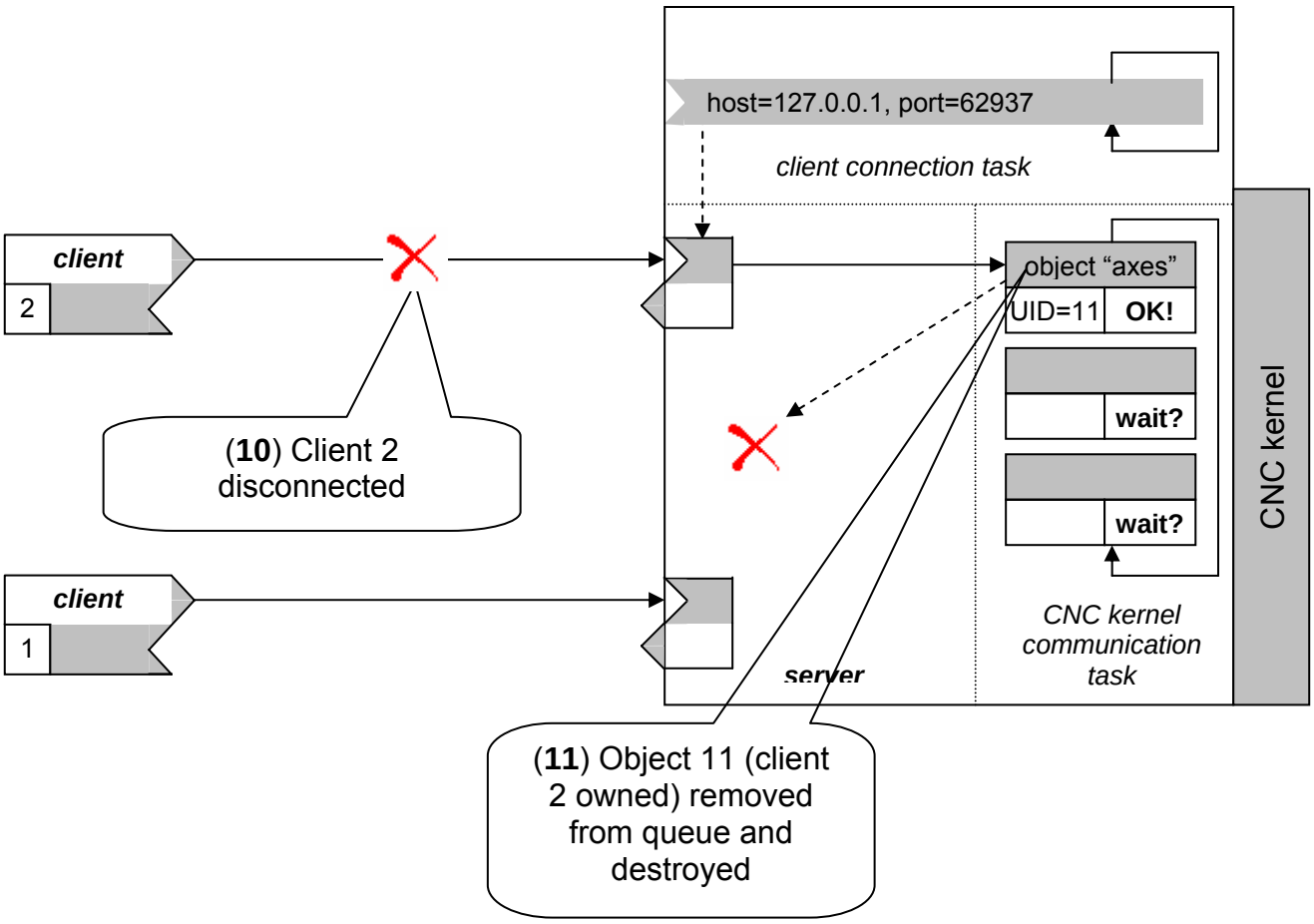
Client can cancel any own request in any time by command **<cancel>**. Normally client doesn't need to add **<uid>** tag in reguest. But client can. In current example the client 2 uses **<uid>11</uid>** in request. Server marks object "axes" (created on this request) by that identifier. If client sends new request in same socket

<cancel><req>yes</req><id>11<id></cancel>

then communication object "axes" is deleted inside server. Server doesn't request more axes position from CNC kernel, because corresponding client's communication object deleted.

Next diagram shows, that communication objects (client owned) are deleted when socket connection broken. So client doesn't need to "care" about corresponding objects in server. All objects will be deleted automatically, when client disconnected from server. Above information is important only for requests with automatic updating mode.

*Subject for technical modifications and errors



Communication Between HMI and User Applets

When user applet is Java then HMI can’t exchange data with applet directly. To provide compatible interface between user applets and HMI for all kind of applets (ActiveX and Java) a communication object **<pool>** is implemented. Because each applet as well as HMI has communication with CNC server, they can contact each to other through server. One applet or HMI can send some information to server by **<pool>**, then other clients can get same information from server. Server doesn’t transfer such information to CNC. Server only saves **<pool>** data into internal buffers to share such information between clients of server.

There are two buffer in server:

- Constant buffer
- Instant buffer

Data is stored in constant buffer when request has **<const>yes</const>** tag. Information is saved in instant buffer in all other cases. Constant buffer’s data exists all time when server is running. Each new client can get complete information from constant buffer, when it sends **<pool>** request. When new information comes to instant buffer then data

can be available for clients which are already active in same time with **<pool>** request. Instant buffer is cleared each time after information is broadcasted between active clients.

Information is stored as identifier, command and value. Some commands and values are reserved for communication between HMI and user applets (see description below). Applets can use own commands (non-reserved!) and any values for communication.

HMI sends following information to constant buffer of pool if corresponding mode is activated in section **[master]** of ini file (i.e. hmi.ini):

- If **level=1**, then HMI sends **<cmd>level</cmd><val>*XXX*<val>**, when actual level changed. Here ***XXX*** is actual level of HMI (decimal format). Highest four bits of double word (32 bits) represent hard key level (0 – level of first hard key, 1 – level of second hard key, …). Next four bits correspond first sublevel of soft keys (1 - first soft key, 2 – second soft key, …) etc.
- If **language=1**, then HMI sends **<cmd>language</cmd><val>*XX*<val>**, when actual language initialized or changed. Here ***XX*** is actual language identifier (**en, de, ru** etc)
- If **station=1**, then HMI sends **<cmd>station</cmd><val>*ncX*<val>**, when actual active station initialized or changed. Here ***ncX*** is actual active station (**nc1, nc2, nc3**, etc)
- If **appletfont=1**, then HMI sends **<cmd>font</cmd><val>*font_name*<val>** during initialization. Here ***font_name*** is actual active font as string

HMI sends following information to instant buffer of pool if corresponding mode is activated in section **[master]** of ini file (i.e. hmi.ini):

- If **userkey=1**, then HMI sends message to pool **<cmd>userkey#*key_name*</cmd>**, when soft key with function **userkey#*key_name*** is pressed. Function of soft key is specified by configuration file (i.e. hmi.cfg). Here ***key_name*** is any name defined by user
- If **softkey=1**, then HMI sends message to pool **<cmd>softkey#X</cmd>**, when soft key is pressed. Here ***X*** is soft key number (1…8).
- If **hardkey=1**, then HMI sends message to pool **<cmd>hardkey#X</cmd>**, when hard key is pressed. Here ***X*** is hard key number (1…8).

So if user applet sends request with automatic updating mode at startup:

“<pool><req>yes</req>_{get}</pool>\n”

then applet will get complete information about actual font, language, station. Additionally applet will get messages when user, soft and hard keys will be pressed (if corresponding HMI mode is specified through ini file). Messages **<pool>** from server contains same commands and values which HMI sent.

HMI interprets following predefined commands from pool, if corresponding mode is activated via hmi.ini file (section **[slave]**):

- If **userkey=1**, then HMI will change status of soft key, specified in configuration (i.e. hmi.cfg) file as **userkey#*key_name***. Here ***key_name*** is any name defined by user.

Any user applet can “ask” HMI to change status of user soft key. To do that an applet has to send corresponding command and value to pool. If applet would like to set title of user key it can send **userkey#*key_name*#title** as command (tag **cmd**) and title text as value (tag **val**). If applet would like to set tool tip of user key it can send **userkey#*key_name*#tooltip** as command (tag **cmd**) and tool tip text as value (tag **val**). If applet would like to set icon of user key it can send **userkey#*key_name*#*icon*** as command (tag **cmd**) and icon as value (tag **val**). Default image file name extension and icon size can be missing in definition of icon. If applet would like to disable user key it can send **userkey#*key_name*#enabled** as command (tag **cmd**) and **off** as value (tag **val**). If applet would like to enable user key it can send **userkey#*key_name*#enabled** as command (tag **cmd**) and **on** as value (tag **val**). If applet would like to select user key it can send **userkey#*key_name*#selected** as command (tag **cmd**) and **on** as value (tag **val**). If applet would like to switch user key in normal state it can send **userkey#*key_name*#selected** as command (tag **cmd**) and **off** as value (tag **val**).

- Applet can set value of user data object. Value of data object may be used in configuration file for specification of view on corresponding level (see SD002EN HMI Setup and Configuration Guide). Client can sends message to pool **<cmd>userdata#*data_name*</cmd><val>*XXX*</val>**. Here ***data_name*** is any name defined by user and ***XXX*** is value to be set for data object by HMI
- If **refresh=1**, then HMI will refresh actual view, specified in configuration (i.e. hmi.cfg), when client sends **<cmd>refresh</cmd>**. So client can change value of user data object and sends refresh to HMI, then layout of applets on current level may be reloaded if views depends on user data object. **NOTE. THIS FUNCTION IS OBSOLET. USE <cmd>macro</cmd><val>refresh</val> instead.**
- If **macro=1**, then HMI will execute macro command that will be send with **<cmd>macro</cmd><val>element[:plugin][@mainplugin]</val]**, here **element** is the element name, **plugin** – the name of plugin that contains element (if empty – main plugin used), and **mainplugin**, if specified, means that only applications with this main plugin will handle this macro command.

Filters can be applied to reduce communication traffic through CNC server. There are two kinds of filters:

- Identifier
- Command

Any client can add **<id>** tag in **<pool>** when **_{set}** is requested. That identifier (digital number) will be associated with command (**<cmd>**). HMI application will add **<id>** for predefined command, if parameter **id** > 0 is specified in section **[master]** of ini file.

When client (applet or HMI) sends a request to get **<pool>** data, it can specify identifier filter by same **<id>** tag. For example **_{get}<id>1-7,10</id>**. In this case client will get only those saved commands from pool which identifiers was not defined (0) or 1, 2, 3, 4, 5, 6, 7, 10. When client sends a request to get **<pool>** data without **<id>** (or 0), then client will get all commands from pool. Identifier filter for HMI is specified by **id parameter** in section **[slave]** of ini file.

*Subject for technical modifications and errors

The similar way is for **<cmd>** filter. When client (applet or HMI) sends a request to get **<pool>** data, then client can specify several command templates, divided by comma. Server will inform client by only those commands, which contains one of templates from the beginning. If command filter is empty, then all commands will be sent to client. For example, if client specified **_{get}<cmd>userkey</cmd>** as filter, then server will sends following commands from pool:

- **userkey#mykey1**
- **userkey#mykey2**

But client will never get information about **level**, **language** from pool. If client would like to get all commands from pool, it has to omit **<cmd>** filter in request.

Both filters (**<id>** and **<cmd>**) are acceptable in one request.

Java Applets

User can develop Java applet and put it in viewer on any level. Java applets can be shown by "**web**" viewer or by "**web**" applet (among other Qt applets in "plain" viewer). User HMI example demonstrate Java applets when soft key 4 "Java applets" selected.

Layout of Java applets on corresponding level is specified by **java.htm** file. Java applets are shown inside HMI standard "**web**" applets. Specification of Java applet is provided by two steps:

- Applet "**web**" gets name of HTML file to be shown in browser by parameter like **<PARAM name='file' value='ddeapplet.htm'/>**
- File **ddeapplet.htm** contains specification of Java class **DDEApplet.class** from Java archive (see HTML documentation for details)

Sun Java vs. Microsoft Java (Windows version only)

Due to known opposition between Sun and Microsoft in Java technology it is recommended to process HMI built-in Java applets by original Sun virtual Java machine, but not Microsoft!

Let demonstrate a difference by "Hello World" example. To display simple "Hello World" applet by MSIE, we need **helloworld.htm** file like following:

```
<html>
<body>
Hello World!
<applet code="HelloWorld.class" width=200 height=40>
    </applet>
</body>
</html>
```

Using of above HTML file as parameter of HMI applet "**web**"
<PARAM name='file' value= helloworld.htm'/>
may cause a fault, because MSIE ActiveX may process **HelloWorld.class** by Microsoft Java machine.

To be sure that original Sun Java machine is applied, we have to transfer **helloworld.htm** to following:

```
<html>
<body>
Hello World!
<!--"CONVERTED_APPLET"-->
<!-- HTML CONVERTER -->
<OBJECT
   classid = "clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
   codebase = "http://java.sun.com/products/plugin/autodl/jinstall-1_4-windows-
i586.cab#Version=1,4,0,0"
   WIDTH = 200 HEIGHT = 40 >
   <PARAM NAME = CODE VALUE = "HelloWorld.class" >
   <PARAM NAME = "type" VALUE = "application/x-java-applet;version=1.4">
   <PARAM NAME = "scriptable" VALUE = "false">
   <COMMENT>
        <EMBED
         type = "application/x-java-applet;version=1.4"
         CODE = "HelloWorld.class"
         WIDTH = 200
         HEIGHT = 40
           scriptable = false
           pluginspage = "http://java.sun.com/products/plugin/index.html#download">
           <NOEMBED>
         </NOEMBED>
        </EMBED>
   </COMMENT>
</OBJECT>
<!--
<APPLET CODE = "HelloWorld.class" WIDTH = 200 HEIGHT = 40>
</APPLET>
-->
<!--"END_CONVERTED_APPLET"-->
</body>
</html>
```

Fortunately Sun provides a special SDK utility (**HtmlConverter.exe**) to make above conversion automatically. See Java SDK documentation for details. You can find **HtmlConverter.exe** utility in **bin** directory of installed Java SDK.

Java Sources

Sources are represented by following files:

- **DataObject.java** implements parsing of xml strings. Such strings are used for communication with CNC server
- **DataThread.java** implements communication with CNC server by socket
- **CPApplet.java** reads/writes cycle parameter P10 from/to CNC. Communication objects <enter>, <select>, <record> are used for that. Be careful to use <record>

because of synchronization problems when several clients try to get data in same time. For example, to get cycle parameter P10 each client has to issue three requests:
<select>_{par}<name>1</name><level>file</level></select>,
<record>_{par}<bof>yes</bof></record>,
<record>_{par}<pos>9</pos></record>. First request selects data for operation. Second request sets pointer at the beginning of file. Last request reads 9th record after first record in the file (P10). The problem may occur if another client sends similar requests to server in same time but for getting of NC part program. As result the first client can get NC block instead of P-parameter. There is no problem for HMI, because HMI doesn't use **<record>**. There will be no problem for OEM client, if only one client will use **<record>** in time. Instead of **<record>** it is recommended to use extended **<file>** communication object with additional parameter **<pos>**. Here we can specify list of records which we would like to get from CNC. For example, server returns only records 1 (first), 3, 4, 5 and 100 on request **<file>_{par}<pos>1,3-5,100</pos><name>1</name></file>**. It is guaranteed that other commands <file>, <select>, <record> will be delayed until current <file> is processed!

- **TestApplet.java** displays actual NC blocks. Applets gets NC blocks by communication object **<blocks>** with automatic updating mode. There are two buttons are located inside applet. Buttons are available when test mode is active. Applet can start/stop NC program processing in test mode by sending single communication object **<mode>**
- **DdeApplet.java** implements DDE interface to PLC. DDE communication in Java is Microsoft specific technology. To get DDE an applet needs special Dynamic Linked Library (DDL). See Java SDK documentation for details. Building of library can be done by Microsoft Visual Studio project **plctest.dsw**, located in **java-dde** folder. Appropriate header file **DDEApplet.h** is produced during compilation of Java sources (see below). Note: be sure that **Project Settings->C/C++->Preprocessor->Additional include directories** for **plctest** project havy correct path for Java SDK includes: **C:\j2sdk1.4.1_05\include,C:\j2sdk1.4.1_05\include\win32**
- **MacroApplet.java** let to select element from list or type it by user and send macro command by <pool> communication object.

Realization of Command Interface to CNC Server

Command interface means that communication objects with single updating mode are used to get or set some CNC data or change CNC mode. Clients sends request to CNC server, waits CNC server's reply in same thread, then client can continue. See realization of command interface in **CPApplet.java**.

At first command socket is defined:

private Socket commandSocket = null;

And two objects **BufferedReader** and **OutputStreamWriter** to read data line by line and write requests.

Initializing of command interface is provides by method:

protected boolean initCommandInterface()

In this method a socket is opened by

commandSocket = new Socket(HOST, PORT);

Then objects **BufferedReader** and **OutputStreamWriter** are connected to

```
in = new BufferedReader(
        new InputStreamReader(commandSocket.getInputStream(),
        "ASCII"));
out = new OutputStreamWriter(commandSocket.getOutputStream(), "ASCII");
```

If error occurs during initialization, then socket closed and method returns **false**.

It is prohibited to call **initCommandInterface()** into **init()** or **start()** methods of applet class, because it may occur security exception in Microsoft Internet Explorer (see “PRB: SecurityExceptionEx Exception Running a Java Applet (Q175622)”). It is recommended in our examples to call **initCommandInterface()** into **getDataRequest()**.

When applet destroyed it must close command socket by method

synchronized protected void closeCommandInterface()

Note: method should be declared as **synchronized**!

Main method of command interface

public DataObject getData(String command)

returns getting data as **DataObject**. Here **command** is XML request. **DataObject** derived from **Properties**. Inside of method we process following steps:

Clear input strem by

```
while (in.ready())
    in.skip(1);
```

Add station to XML request by

```
int n = command.indexOf('>');
command = command.substring(0, n + 1) +
    "<st>" + parameters.getProperty("station") + "</st>" +
    command.substring(n + 1);
```

Send request to socket by

```
out.write(command + '\n');
out.flush();
```

Wait answer by

```
String str = in.readLine();
```

Parser reply by

```
data = new DataObject();
data.fromXML(str);
```

Check error by

```
String err = data.getProperty("err");
if (err != null && !err.equals("ok"))
    data = null;
```

If error found, then return **null**.

Other methods

public boolean sendCommand(String command)

public String getStringData(String command, String dataName)
public int getIntData(String command, String dataName)
was implemented to simplify frequently used operations.

Getting of Data From CNC Server on Request With Automatic Updating Mode

TestApplet.java example uses **<blocks>** communication object with automatic updating mode to get actual NC blocks. Normally client sends similar requests to server, but can't wait replay from server in same thread, because server will informs client asynchronously when new data will be available from CNC kernel.
Java socket allows synchronous communication only. So socket should be opened in independent thread to provide asynchronous data exchange.

Implementation of asynchronous socket communication is located in special class **DataThread** (see **DataThread.java**) derived from **Thread**.

Applet creates asynchronous communication channel in method **start()** (see **TestApplet.java**):

```
parserThread = new DataThread(this);
```

and calls

```
parserThread.resumeConversation();
```

to create communication thread.

Thread creates a socket by

```
socket = new Socket(component.host, component.port);
```

See **run()** method in **DataThread.java**.
Then thread gets XML requests from applet:

```
String[] dataRequest =  component.getDataRequest();
```

And transfers requests to server via socket:

```
for (int i = 0; i < dataRequest.length; i++)
{
        writeLine(dataRequest[i]);
}
```

Thread waits data from server in loop:

```
str = in.readLine();
```

Parsers getting XML strings:

```
DataObject data = new DataObject();
data.fromXML(str);
```

Finally it sends results to applet:

```
component.dataReady(data);
```

How to Build Java Archive

User HMI application uses Java applets from archive **user.jar**. Java sources to build archive are located in **java** folder. User can rebuild archive by following steps:

- Download Java SDK j2sdk-1_4_1_05-windows-i586.exec (or higher) from java.sun.com

*Subject for technical modifications and errors

- Install Java in folder C:\j2sdk1.4.1_05. If installation done into other folder or new version of SDK is used, then **JDIR** environment variable should be changed on new path (see **java-build.bat**)
- Run **java-build.bat**

Next actions are carried out by **java-build.bat** (see SDK documentation for details)**:**

- Compiling of all *.java files
- Building of header file **DDEApplet.h** to create additional library **plctest.dll.** Above library provides Java applet by DDE interface (Windows only specific). Copying of header file to **java-dde** folder
- Building of **user.jar** archive included all applets
- Creating of store and key
- Creating of certificate. Procedure requires some user's information by user. Meaningless input is allowed because that is only test example
- Signing of archive by certificate

Development of ActiveX Controls

It looks like development of Java applets but it is based on Microsoft ActiveX technology which are available on Windows platform only. ActiveX controls can be produced by Visual C++, Visual Basic, etc. Most of toolkits by Microsoft are commercial.

Implementation of ActiveX control on layout (view) is possible in two ways:

- ActiveX control can be specified on HTML page and shown in active view via HMI applet **web** or **web** viewer in same manner as for Java applet. Such method is not recommended due to focus problems (see FAQ below)
- It is recommended to use standard HMI applet **activex** to implement an ActiveX control directly in HMI layout. See **activex.htm** source in **userhmi\xml** as example

HMI Standard Applet "activex" as Container of Control

Example of xml code for specification of ActiveX control in view by **activex** HMI applet is shown below:

```
<APPLET    code='activex'
           left='0'
           top='10'
           width='8'
           height='3'>
<PARAM name='id' value='OEM'/>
<PARAM name='foreground' value='FF0000'/>
<PARAM name='title' value='1'/>
<PARAM name='titletext' value='ParamX'/>
<PARAM name='margin' value='0'/>
<PARAM name='border-width' value='1'/>
<PARAM name='roundness' value='0'/>
<PARAM name='stretchable' value='1'/>
<PARAM name='clsid' value='C5847FCD-8B38-4642-8479-D15174731656'/>
```

Specific parameter of applet is **clsid**. Class identifier of ActiveX control (CLSID) is reported here. Each control gets unique CLSID on designing stage.

Control gets values of xml parameters as properties (see **ParamX** project for details of realization):

- **foreground** as standard property **ForeColor**
- **background** as standard property **BackColor**
- **font** as standard property **Font**
- **lines** as property **lines**
- **port** as property **port**
- **id** as property **id**
- **stretchable** as property **stretchable**
- **time** as property **time**
- **host** as property **host**
- **station** as property **station**

Note: It is necessary to register ActiveX controls on machine before using them in HMI application. Registration is carried out by special utility **regsvr32.exe:**

regsvr32.exe /s /c activex\ParamX\ParamX.ocx

See **userhmi.bat** file where registration done for all examples of controls.

ActiveX Examples

User HMI are provided by ActiveX examples. ActiveX controls are shown among other standard HMI applets when user selects soft key 5.
Note: running of User HMI needs CNC kernel and CNC server (user can start standard HMI), because examples use communication with kernel.

Each example is represented as C++ Microsoft Visual Studio project. There are following projects:

- **CPAppletX** demonstrates dialog box to get and set cycle parameter P10. In contrast to Java applet **CPApplet** the ActiveX control uses extended communication object **<file>**, to get only one record from file in safe mode. Safe mode guarantees that processing of **<file>** request will lock CNC server for other similar requests. Same communication object **<enter>** is used to set value of parameter P10. Control shows also as input focus transferred from HMI frame (soft keys, status window) to ActiveX and back. Use **Tab** key to move a input focus, when mouse is not installed. In some cases an user must press **Tab** several times to get input focus in required window
- **KeyAppletX** demonstrates handling of user soft keys by ActiveX. Two keys with **userkey** functions are specified on level of ActiveX examples. Because of parameter **userkey=1** (see **[master]** and **[slave]** sections of **user.ini** file) a User HMI application informs applets via command **<pool>**, when corresponding soft key pressed. In same manner an applet can send new status, title, tool tip and icon for user soft key by communication object **<pool>**. User HMI will overwrite status, title, tool tip and icon of key

- **NCStateX** demonstrates consistency of OEM ActiveX control among other HMI standard applets, because it gets same parameters (font, colors, etc) from frame. Note: value of **border-width** parameter of applet have been set to 2 (see **activex.htm**). Control is shown in right upper corner of layout. **NCStateX** uses communication object **<info>** to get state of NC kernel. Standard HMI applet **ncstate** uses **<ncda><var>status</var></ncda>** request. Using of **<info>** gives more possibilities to display right state of CNC kernel (i.e. problem with Drive ON missing display)
- **ParamX** demonstrates how ActiveX control gets HMI settings via properties. Additionally the control demonstrates communication procedure with HMI kernel, when ActiveX is closed. If view's permanent property is off (**<BODY ' permanent='0'** …), then HMI kernel deletes all applets from actual view, when view has to be changed (i.e. HMI level changed). Even view's permanent property is on (**<BODY ' permanent='1'** …) and applets stay in memory, when view has to be changed, nevertheless HMI kernel deletes all applets, when HMI application closed. During closing procedure a HMI kernel calls special method **BOOL checkForClose()** for Qt applets and ActiveX controls. When **checkForClose()** method returns **TRUE**, then HMI may close ActiveX. When **checkForClose()** method returns **FALSE**, then HMI breaks closing procedure. See **activex\ParamX\ParamXCtl.cpp** for details

Development User Specific Objects (Applets, Functions) by Qt Toolkit

When user requires applets and functions for specific technology, new components can be developed by Qt Toolkit (http://www.trolltech.com/). The same toolkit is used for development of standard HMI (kernel and plug-ins). User can order development from PA/PA Soft or design new Qt modules himself. In last case user needs Qt Toolkit by Trolltech (non-commercial version is free for Linux) and C++ compiler (for example, VC++ for Windows). Internal program interface for HMI Qt applets will be specified later by PA Soft. Interface to CNC is open and already documented. When new applet built, it can be integrated into standard HMI without problem. Integration is carried out via XML and configuration files.

Unfortunately such solution might be complicated for user, because user needs special knowledge in HMI internal program interface. But this way is very powerful.

Not only applets but other HMI specific object can be designed by this method. New objects are united in Dynamic Link Library (plug-in). Additional DLL doesn't influence on other standard HMI plug-ins.

Objects are compatible for Windows and Linux versions on source code level.

Others

It means a development of specific HMI by any other technology which is well known for user. It is only requested that programming tool supports socket. There aren't other restrictions because communication to NC kernel is carried out by open XML based protocol.

Note. As CNC server provides multiply client connection, user specific HMI can run simultaneously with standard HMI.

*Subject for technical modifications and errors

Comparison of Technologies

Method	Functionality	Project time	Complexity	Development toolkit	OS compatibility
Configuration of standard HMI	Inside standard HMI functionality	Short	Low	No additional toolkit	Windows Linux
Development new extensions on C++ included Qt library	High	Middle	High Support by PA/PA Soft needed	Qt Toolkit, C++ compiler (commercial for Windows	Windows Linux
Using of Java applets inside HMI	Middle	Middle	Middle	Free Java	Windows
Using of Java applets on DHTML pages for browsers (without HMI frame)	Middle	Long	High for complicated application	Free Java	Windows Linux
Using of ActiveX controls inside HMI	Middle	Middle	Middle	Commercial VC++	Windows
Using of ActiveX controls on DHTML pages for browsers (without HMI frame)	Middle	Long	High for complicated application	Free Java	Windows
Other user programming technology based on open CNC server interface	High	Long	High	Depends on user technology	Windows Linux

FAQ

How Many Soft Key Levels Are Possible?

Maximum 7 levels can be configured directly. Using of complicated functions (i.e. Editor of machine parameters into HMI, so called built-in application) allows to get unlimited number of levels indirectly.

What's Happened With The Focus Control In Windows, When Will It Change From HMI To Applets?

User can see focus problems with Java applets. It depended on versions of Microsoft IE and Java VM. Java applet can get input focus when it becomes visible in layout, but it can't transfer focus back to HMI by keyboard (Tab). Handling of hard and soft keys are locked in that case. There is no problems with mouse operation.

Using **activex** Qt applet solves focus problems for Windows based control. New applet is container of ActiveX control. Now if ActiveX control needs input focus, then it can get focus in HMI in usual manner (by Tab). There is no more problem to switch input focus between HMI applets (status window and so on) and integrated ActiveX controls on current level.

Can Java Applets or ActiveX Access The Soft Keys?

Applet (ActiveX or Java) can get events when hard and soft key pressed (possibility is specified by ini file). One of ActivX example shows how to get event when soft key pressed and set soft key status and text. Communication object <pool> is used to exchange information between HMI and applets. If applet needs operator input (not SK or HK), it has to get input focus by mouse or Tab.

How Can An OEM Applet (Java, ActiveX) Work With Selected HMI Fonts, Colors?

There are two possibility to transfer information to ActiveX controls and only one way for Java:

- When applet **activex** is created (becomes visible) on actual HMI level, it transfers HMI system information to ActiveX control. Font, background and foreground colors (see specification of applet parameters in “SD002EN HMI Setup and Configuration Guide”) are transferred to ActiveX control via standard properties: **Font**, **BackColor**, **ForeColor**. To get other applet’s parameters (lines, stretchable, host, port, time, station, id) an ActiveX control must implement properties with same name.
- Java applets and ActiveX controls (in browser) can get font, language by <pool> communication object

I search for a possibility to get soft key information in an additional applet

Applets can operate only with SKs assigned as "userkey" in configuration file (see user.cfg from userhmi example). Let applet A1operates with SK6 (50000006 = userkey#activexkey2).

When A1 wants to select softkey, it sends usually messages to instant pool:
<pool>_{set}<cmd>userkey#activexkey2#selected</cmd><val>on</val></pool>

When A1 wants to make softkey as normal, it sends usually:
<pool>_{set}<cmd>userkey#activexkey2#selected</cmd><val>off</val></pool>

For example, applet A2 is located on same level. Applet can issue
<pool>_{get}<auto>yes</auto></pool>

*Subject for technical modifications and errors

on initialising phase. Then applet A2 will get all messages from pool including messages of applet A1 to select or deselect SK6. Something like this:
<pool>_{get}<auto>yes</auto><cmd>userkey#activexkey2#selected</cmd><val>off</val></pool>
By this message applet A2 can check state of SK6.

If applet A3 is located on other level, then unfortunately A3 can't get state of SK6 by getting of instant pool, because A3 doesn't exist when A1 is active. In this case we must use constant pool. To send message to constant pool A1 has to prepare XML string like following:
<pool>_{set}<cmd>userkey#activexkey2#selected</cmd><val>on</val><const>yes</const></pool>
Now when A3 will start, then it get messages about selection of SK6 by A1, because message is in constant pool.

A3 can avoid useless messages from pool by filter. For example, A3 can query pool messages by
<pool>_{get}<auto>yes</auto><cmd>userkey#activexkey2#selected</cmd></pool>
Now A3 will get only messages for userkey#activexkey2 (SK6) concerning selection or deselection of soft key. Of course, you can use another filters (i.e. by <id> of messages).

Java applet is shown by Microsoft Internet Explorer but not displayed inside HMI

User report: I installed the Sun JDK 1.4.2_02 and Java runtime environment. I have made a small test Java Applet. It's made by JDK 1.4.2_02. I can see it in Internet Explorer 6.0, but I can't see it in HMI.
Solution: Possible reason is that HMI applet **web** uses Microsoft virtual Java machine instead of original Sun JVM to process Java applet. See **Sun Java vs. Microsoft Java** for details.